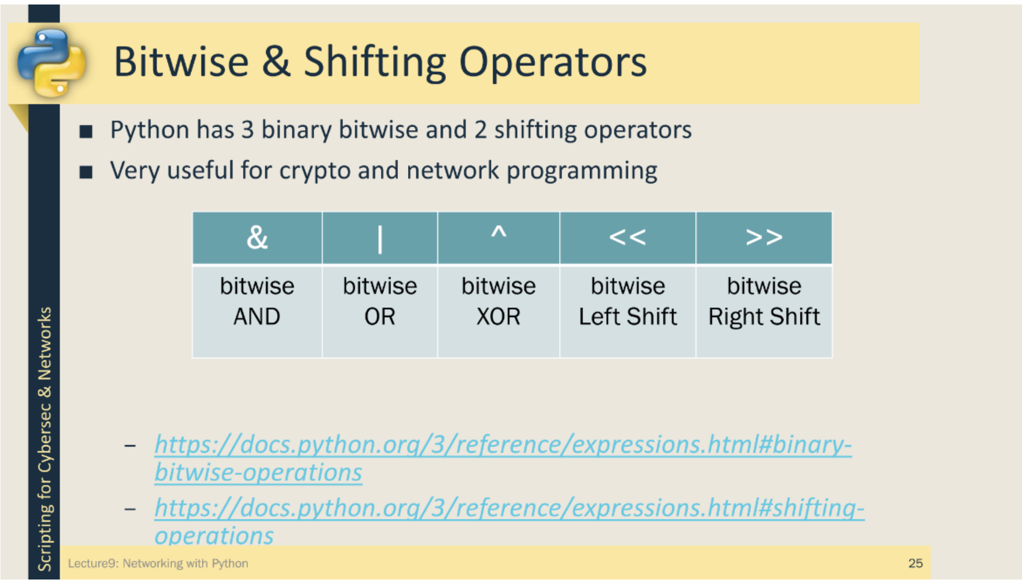
Lab 9 optional extension:
Manual decoding of IP addresses

In the lab, Exercise 2.1.4, you used socket.inet_ntoa() to decode the IP addresses from binary format.

In this optional extension, you will learn from first principles how the process actually works, and write your own decoding function.

1 Background information

The slides below introduce the bitwise and shifting operators that we will use in tis exercise.



Bitwise & Shifting Operators

- Python has 3 binary bitwise and 2 shifting operators
- Very useful for crypto and network programming

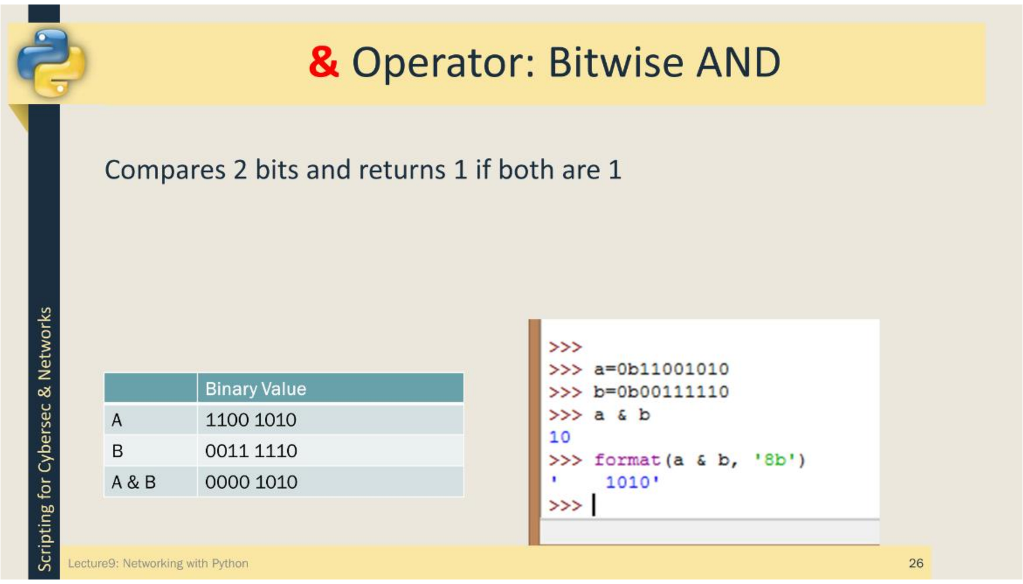
&	|	^	<<	>>
bitwise AND	bitwise OR	bitwise XOR	bitwise Left Shift	bitwise Right Shift

– https://docs.python.org/3/reference/expressions.html#binary-bitwise-operations

– https://docs.python.org/3/reference/expressions.html#shifting-operations

Scripting for Cybersec & Networks

Lecture9: Networking with Python 25



& Operator: Bitwise AND

Compares 2 bits and returns 1 if both are 1

	Binary Value
A	1100 1010
B	0011 1110
A & B	0000 1010

```
>>>
>>> a=0b11001010
>>> b=0b00111110
>>> a & b
10
>>> format(a & b, '8b')
'    1010'
>>> |
```

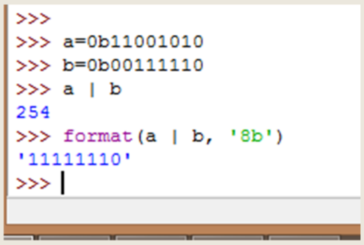
Scripting for Cybersec & Networks

Lecture9: Networking with Python 26

| Operator: bitwise OR

- Compares 2 bits and returns 1 if either are 1

	Binary value
A	1100 1010
B	0011 1110
A | B	1111 1110

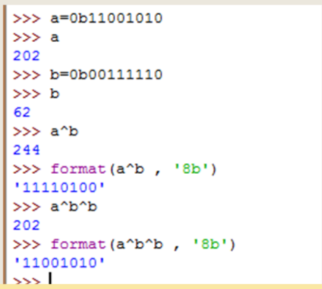


```
>>>
>>> a=0b11001010
>>> b=0b00111110
>>> a | b
254
>>> format(a | b, '8b')
'11111110'
>>> 
```

^ Operator: bitwise XOR

- Compares 2 bits and returns 1 if either are 1, but not both
- Xor'ing 'a' with another number twice, gets back 'a'

	Binary Value	Decimal Value
A	1100 1010	202
B	0011 1110	62
A ^ B	1111 0100	244
A ^ B ^ B	1100 1010	202

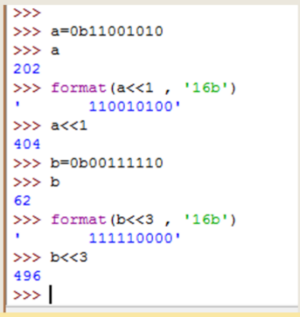


```
>>> a=0b11001010
>>> a
202
>>> b=0b00111110
>>> b
62
>>> a^b
244
>>> format(a^b , '8b')
'11110100'
>>> a^b^b
202
>>> format(a^b^b , '8b')
'11001010'
>>> 
```

<< Operator: left shift

- Shifts bits to the left by the given number of places
- The lower bits are filled with zeros
- Each left shift multiplies by 2

	Binary	Decimal
A	1100 1010	202
A << 1	1 1001 0100	404
B	0011 1110	62
B << 3	001 1111 0000	496

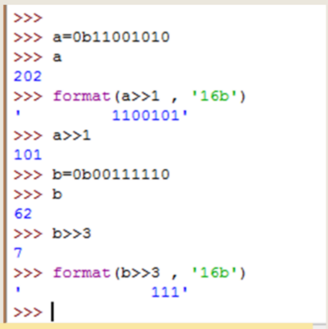


```
>>>
>>> a=0b11001010
>>> a
202
>>> format(a<<1 , '16b')
'       110010100'
>>> a<<1
404
>>> b=0b00111110
>>> b
62
>>> format(b<<3 , '16b')
'       111110000'
>>> b<<3
496
>>> 
```

>> Operator: right shift

- Shifts the bits to the right by a given number of places
- The upper bits are filled with zeros
- The lower bits are discarded
- Each right shift divides by 2

	Binary	Decimal
A	1100 1010	202
A >> 1	0110 0101	101
B	0011 1110	62
B >> 3	0000 0111	7



```
>>>
>>> a=0b11001010
>>> a
202
>>> format(a>>1 , '16b')
'         1100101'
>>> a>>1
101
>>> b=0b00111110
>>> b
62
>>> b>>3
7
>>> format(b>>3 , '16b')
'             111'
>>> |
```

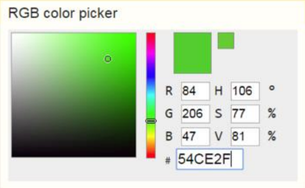
Bitwise and Shifting Operators example: RGB Colours

- Every RGB Colour is represented by a 3 byte number
- 1 byte used for each of R, G & B
- Example: Hex 54CE2F

R G B
84 206 47
decimal

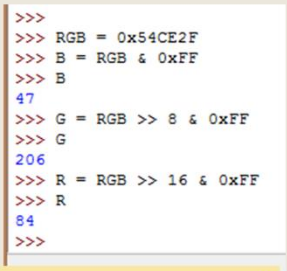
- Use mask (&) and shift (>>) to obtain individual values for R, G and B

http://www.rapidtables.com/web/color/RGB_Color.htm



RGB color picker

R 84 H 106 °
G 206 S 77 %
B 47 V 81 %
54CE2F



```
>>>
>>> RGB = 0x54CE2F
>>> B = RGB & 0xFF
>>> B
47
>>> G = RGB >> 8 & 0xFF
>>> G
206
>>> R = RGB >> 16 & 0xFF
>>> R
84
>>>
```

1.1.1 Reformat IP addresses into standard dotted decimal format

ip addresses are generally shown in this format: ‘127.0.0.1’. But in the pcap record they’re a 32 bit binary streams e.g. *src=b'\x92\xb0\xa4[', dst=b'\x17\x156\x83'*. Each byte (8 bits) represents one of the octets in the dotted decimal format.

e.g b'\x92\xb0\xa4[' is 146.176.164.91

because: \x92 = 146 decimal; \xb0 = 176 decimal; \xa4 = 164 decimal and [is 91 decimal.

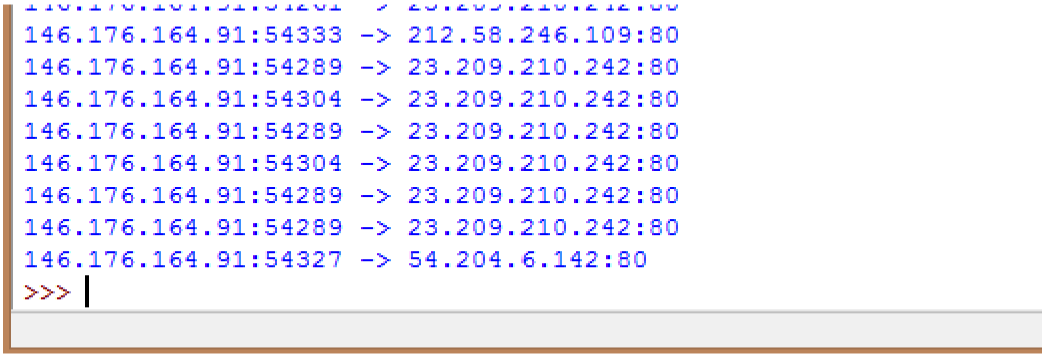
Create a new function called decode_ip_v2() which will take an ip address as a binary stream and return the ip address in dotted decimal notation. Use the code below as a starter and refer to the lecture notes on Bitwise operations, specifically the example for RGB colours, for help. Remember to change your print statement in main() to call decode_ip_v2 for the src and dest ip addresses.

```
def decode_ip_v2(orig_ip):
    # convert orig_ip to an integer
    pass

    # construct decoded_ip as a dotted decimal string using '>>' and '&'
    decoded_ip = ''

    return decoded_ip
```

The output should now look similar to this….



```
146.176.164.91:54333 -> 212.58.246.109:80
146.176.164.91:54289 -> 23.209.210.242:80
146.176.164.91:54304 -> 23.209.210.242:80
146.176.164.91:54289 -> 23.209.210.242:80
146.176.164.91:54304 -> 23.209.210.242:80
146.176.164.91:54289 -> 23.209.210.242:80
146.176.164.91:54289 -> 23.209.210.242:80
146.176.164.91:54327 -> 54.204.6.142:80
>>> |
```

1.1.2 Reformat IP address using a Loop or List Comprehension - Challenge Question

If you’ve not already done so, reformat the ip address using a loop. Once that’s done, rewrite the loop into a list comprehension. List comprehensions are difficult to become familiar with but can be more easily written if an intermediate looping step is taken. For example:

```
# Looping method
# build an array of octets then join them using ','.join()
octets = []
for i in Looping_Criteria:
    octets.append( Code_to_Generate_Next_Octet )
    decoded_ip = '.'.join(octets)

# List comprehension just rearranges these into 2 lines
octets = [ Code_to_Generate_Next_Octet for i in Looping_Criteria ]
decoded_ip = '.'.join(octets)

# Or all in 1 line!!
decoded_ip = '.'.join( [ Code_to_Generate_Next_Octet for i in Looping_Criteria ] )
```